`

Identification Of Malware Using Enhanced Malware Detection Pre-Processing Techniques

[1]Mrs. M .Meena Krithika, [2]Dr. E. Ramadevi

[1]Assistant professor,
[2]Associateprofessor,
[1]Department of computer science,
[2]Department of computer applications,
[1,2] NGM college, Pollachi

Abstract

This research paper proposes a machine learning based malware analysis framework, which is made out of three modules: data processing, decision making, and new malware detection. The data processing module manages text to ASM, Opcode n-gram, and import functions, which are utilized to remove the features of the malware. The decision-making module utilizes the features to group the malware and to identify suspicious malware. Malware designers have been profoundly fruitful in evading the signature based detection techniques. The greater part of the prevailing static analysis techniques involve an instrument to parse the document. The entire analysis process gets dependent to the efficiency of the instrument; if the device crashes the procedure is hampered. The greater part of the dynamic analysis techniques involve the binary document to be run in a sand-boxed environment to examine its behaviour. This can be handily upset by hiding the malicious activities of the _le on the off chance that it is being run inside a virtual environment.

Keywords[*Malware detection, suspicious malware, opcode, ASM*]

1. Introduction

An enormous number of researches have read techniques for analyzing and detecting malware. Traditional business antivirus items for the most part depend on signature-based technique, which needs a nearby signature database to store patterns extricated from malware by specialists. Be that as it may, this methodology has incredible limitations since explicit minor changes to malware can change the signature, so increasingly more malware could without much of a stretch avoid signature-based detection by encrypting, obfuscating, or packing. Identifying malicious software executables is made troublesome by the constant adaptations introduced by miscreants in request to sidestep detection by antivirus programming. Such changes are akin to mutations in organic sequences. Recently, high-throughput techniques for gene sequence classification have been created by the bioinformatics and computational science communities..

The issues with the need to store a huge number of words turns out to be even progressively hazardous when the size of the letter set increases. This is plainly the situation when we consider accumulated code or source code. Strand tends to this issue by utilizing a type of lossy compression called Minhashing which despite everything underpins sequence comparison, however with a much diminished memory footprint.

Advantages and Limitations Since, during dynamic analysis malware is being executed, this will assist us with countering the techniques defeating static analysis, for example, packing and obfuscation. Significant limitations of Dynamic Analysis are that we are generally ready to monitor a single way of execution, so it experiences incomplete code inclusion. Additionally, on the off chance that a malware can identify the sandbox, then it might modify it conduct by hiding the malevolent exercises, along these lines evading the analysis. Another limitation is that, on the off chance that there is any bug in the sandbox environment, then malware can infect the host PC or different PCs on the network.

Malware Nomenclature albeit all anti-infection companies follow thorough naming conventions for malwares however there is no standard conventions for naming malwares. So it is very conceivable that different anti-infection engines may assign different names to the equivalent malware record. With the end goal of this proposition we will examine the naming convention followed by Microsoft anti-infection. Microsoft anti-infection utilizes the malware naming plan proposed by Computer Antivirus Researchers Organization (CARO).

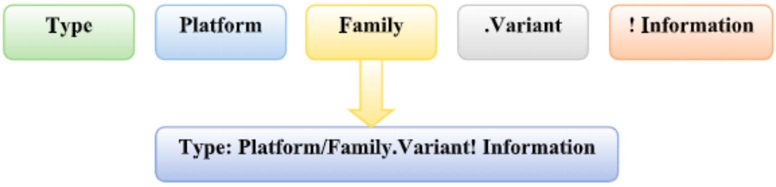

Figure 1: Naming Scheme for Malware

Malware belonging to a similar family have code likenesses, which help in creating generic detection and evacuation techniques. It is the most important field in the naming convention. Variant is utilized to distinguish between different versions of a similar family. Information field is utilized to indicate some additional information about the malware, for example, regardless of whether it is debased, compacted, pressed, and so on. It might likewise include any other information which the analyst considers as an important quality of that malware. For instance link might be added to determine that the malware is an alternate route document. A malware name must include the family name and the variant name different fields might be dropped. Only if their is only one variant of a specific family is known, variant field is permitted to be dropped. At places in this theory family has been supplanted by class, subsequently malware class and malware family are essentially the equivalent.

Prior works on Malware detection

A malware can be in type of a content, an executable or any other software program. Comprehensively, malwares can be named Worms, Viruses, Trojans, Ransomwares, Adwares, Spywares, Bots, PUPs 1 , Rootkits, Scarewares, and different malicious programs. Pretty much every assault we find out about, for example, the Mirai malware assault and WannnaCry Ransomware assault, involves a malicious programming. With the rising advanced footprint the intricacy of these assaults is likewise increasing.

Malware are being utilized to assault basic infrastructures, for espionage against a nation, for stealing private information or for conducting financial fakes. All the assaults use network as a medium. Practically all the malware detection framework in industry use either signature based methodology or anomaly based methodology. A signature is a unique sequence of bytes that is present in the malicious binary and in the records that have been harmed by that malware. Signature based strategies utilize the unique signatures created by the anti-infection companies using the known malware to catch the danger. This methodology is quick and has high precision, however it comes up short in detecting already unseen malware. So generally, after a new malware has infected numerous frameworks an analyst might have the option to generate its signature. Likewise the signature database must be arranged manually which is a tedious procedure. In anomaly based methodology the anti-virus company's frames a database of actions that are considered safe. In the event that a procedure breaks any of these predefined principles, it is named as malicious. In spite of the fact that with anomaly based strategy we can identify new unseen malware tests yet the bogus alert rate is high.

2.Prior Algorithms Used For Malware Detection

1. Logistic Regression:

Logistic Regression is fundamentally a directed characterization algorithm. In a characterization issue, the objective variable (or yield), y, can take simply discrete characteristics for a given arrangement of features (or inputs), X. Rather than prevalent thinking, determined backslide IS a backslide model. The model structures a backslide model to anticipate the probability that a given information section has a spot with the classification numbered as "1". Much equivalent to linear backslide expect that the information seeks after a straight work, Logistic backslide models the information utilizing the sigmoid limit. Logistic regression turns into a request strategy exactly when a decision limit is brought into the picture. The setting of the breaking point regard is a significant piece of Logistic regression and is reliant on the grouping issue itself.

2. Random Forest

Random forests, a mainstream machine learning approach us used to characterize malware and benign applications. Random forest fundamentally is utilized for classification and regression. It consists of set of binary decision trees. Through the training of different decision trees, the algorithms combines results from these trees with voting approach. In the wake of extracting the features, random forest calculation is utilized for classification. The name in the event that we separate the word, it consists of 'forest' which consist of gathering of decision trees, and the word 'random' comes since we are doing random sampling. On applying this calculation on an informational index, it takes a subset of the information as training set and bunches the information into gatherings and subgroups. On connecting the information points to gatherings and sub-bunches we get a structure resembling a tree, called decision tree. The calculation then readies a number of trees, resembling a forest. In any case, each tree is different, concerning each split in the tree, the factors are chosen randomly. The remaining informational index, aside from the training set is utilized for predicting the tree in forest which makes best classification of information points and the tree having most prescient force is shown as yield. Then, a set of lables is set to determine the kind of each application where 1 denotes malware and 0 denotes benign applications. At every node of the decision tree, it parts the training set into two subsets with different marks by minimizing the uncertainty of the class labels.

3. Decision Tree

Decision trees are one of the broadly utilized and useful strategies for inductive inference. It is a strategy for approximating discrete esteemed objective functions in which the learned function is represented by a decision tree. Decision trees give a basic arrangement of decides that can sort new information. Creating decision trees requires a pre-grouped dataset in request for the calculations to learn patterns in the information. This training dataset is comprised of features which are quantifiable qualities of the information. When the decision tree is worked from these features, the principles for characterizing information can be utilized to identify and arrange new information of interest by incorporating the rationale into existing defenses like IDS, firewalls, exclusively constructed detection contents, or classification programming. Decision trees are devices for analyzing information and identifying significant attributes in network information that indicate malicious exercises. Decision trees can help groups to determine which IDS signatures to compose, which firewall rules to implement, and what sort of network action to hail for additional analysis. Be that as it may, decision trees alone don't make a move to stop danger, similar to firewalls and Intrusion Prevention Systems (IPS). Their decision rationale can be utilized in conjunction with other continuous devices to make remedial move against digital dangers by highlighting what the malicious action resembles. In the classification issue, Decision Tree ready to accomplish most noteworthy exactness with 93.3% for multiclass and 94.6% for binary classification. Likewise, the outcome accomplished by Decision Tree is far superior to Sandbox.

4. Adaboost

AdaBoost is one of the most common boosting algorithms in ensemble learning, and is short for Adaptive Boosting. This calculation can be utilized in conjunction with many other machine-learning calculations to improve the detection precision. AdaBoost bolsters a weight distribution over the training set to minimize mistakes and expand the margin regarding the features. It can generate powerful and precise predictions by combining many basic and reasonably exact speculations into a strong theory. AdaBoostM1 is one of the two significant versions of AdaBoost calculations for binary-classification problems.AdaBoost consists of many vulnerable classifiers, and renders the

`

absolute last yield the utilization of weighted vulnerable classifiers. It is a versatile version, on the grounds that a helpless classifier can be refreshed the utilization of misclassified results.

5. Gradient Boosting

Gradient boosting is one of machine learning algorithms utilized for classification and regression. It combines models from different calculations to create new iterative one. Gradient boosting is one of the most uncontrollably utilized machine learning calculations because of its exactness and efficiency. It began with the Adaptive Boosting (AdaBoost) then formed into many calculations and techniques, for example, GBM. It is an ensemble of the model using a boosting technique, essentially combining powerless classifiers to frame a stronger one. In particular, for this situation, the feeble classifiers are different decision trees, hence the consequence of GBDT model could be considered as a combination of these decision trees. The boosting procedure is sequential, which connects the subsequent trees to the past ones with blunders in the predictions generated in the past in request to diminish the prediction mistake continuously during the combination procedure. Also, the GBDT model won't consider the outcome from one single tree as the final outcome. Along these lines, it tends to be utilized to take care of the overfitting issue too.

3.Proposed Pre-Processing For Malware Model

Initial component engineering consisted of extracting different keyword counts from the ASM files just as the entropy and document size from the BYTE documents of the 10868 malware tests in the training set. Image files of the initial 1000 bytes of the ASM and BYTE documents were made and combined with watchword and entropy information. This brought about a lot of 2018 features. Flow control graphs and call graphs were generated for each ASM test. A list of capabilities was then generated from the graphs, including graph greatest delta, density, width and function counts and so forth.

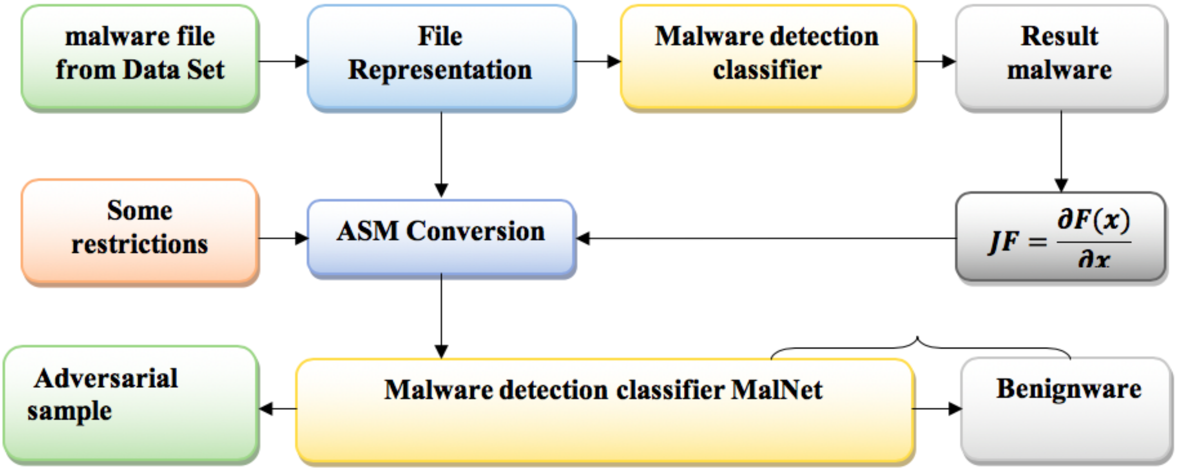
malware file from Data Set
File Representation
Malware detection classifier
Result malware
Some restrictions
ASM Conversion
$JF = \frac{\partial F(x)}{\partial x}$
Adversarial sample
Malware detection classifier MalNet
Benignware

Figure 2: The malware detection

Learning Opcode Sequence through LSTM:

In this section, we introduce another important part of Proposed EMDT, which manages opcode sequence with LSTM to learn malicious sequence features and patterns. Opcode sequences are removed from decompiled files. These sequences really reflect code rationale and program execution rationale of official files. Hence, LSTM can mine malicious code sequence features corresponding to elevated level malicious behavior from them.

Opcode Sequence Extraction:

To learn from opcode sequence, first we need to extricate opcode sequence from raw official fles. We decompile the official file through IDA Pro which generates .asm design decompiled file. IDA Pro is a common decompilation and debugging tool that settle malware into Intel x86 get together instructions.

Then for the .asmfile, we navigate all lines and cut sentences through space character as a delimiter to coordinate each expression to our predefined opcode set which contains all common Intel x86 get together instructions. In the event that the matching is effective, we retain the opcode; else, we erase the expression. During this procedure we find that there are a huge number of copy opcode subsequences on decompiled files, for example, dd,dd, . . . , dd or db, db, db, . . . , db. So it is required to filter these duplicate subsequences by adding a few principles. Te pseudo code of our opcode sequence extraction algorithm is shown below.

Opcode sequence extraction algorithm

```
files = get_files( );
for l in files;
    file = open(l.asm);
    for line in file;
        words = .split(“ ”);
            for word in words;
                The current word belongs to opcode set opcode_set;
                 The last 3 words are not duplicated opcodes;
                   if word in opcode_set and (word!= last_word & last_last_word!= last_word)
                   last_last_word = last_word;
                   last_word = word;
                   filter_words.add(word);
                   endif;
              end for;
      end for;
end for;

Input: Executive file
Output: Opcode sequence
```

During the time spent opcode sequence extraction, the size of the opcode set will affect the average length of opcode sequences. Bigger opcode set will acknowledge more kinds of opcodes. Since there are many noise information and too long opcode sequence will cause difficult learning issue with LSTM, we need to confine the size of the opcode set in a reasonable range so it only contains the most legitimate information. So we treat all decompiled .asmfiles as text and instructions as vocabularies. Then we make frequency statistics and filter out the low frequency vocabularies. After that we utilize every jargon frequency as an element and play out a classifcation by a random forests model, random forests can give the best feature importance. We pick the vocabularies which give the best feature importance. Finally we get opcode set including 185 elements and extract opcode sequences with that. At this point these opcode sequences ought to be digitized before being utilized as input of neural network; we utilize one-hot encoding which just takes a mapping transformation to get a sparse vector like [0, 0, 0, 1, 0, . . . , 0] whose N binary status bits represent N states only containing one nonzero element. And each opcode gets a unique one-hot representation.

Very Long Sequence Learning by LSTM.

We first briefly introduce LSTM network. As a deep neural network Long-Short Term Memory (LSTM) is broadly utilized for processing time arrangement data, which is an improved model based on Recurrent Neural Networks (RNNs). RNN utilizes an internal state to represent past input esteems which permits it to catch worldly context. Based on this, LSTM utilizes the Constant Error Carousel (CEC) and all around designed "door" structures to facilitate the vanishing gradient issue during mistakes back propagation. So misfortune can flow in reverse through longer time step, which enables LSTM to learn long-term dependency and context. In brief, LSTM includes three entryways (input door, overlook door, and yield door) to choose and control the CEC state. Here Proposed EMDT utilizes LSTM network to learn from opcode sequence for malware detection.

Notwithstanding, one existing issue for LSTM is that it is difficult to effectively train when the input sequence is excessively long, however LSTM can catch longer time arrangement context than RNN. In our work, in the event that the size of official fle is extremely enormous, then the length of separated opcode sequence is long. For instance, the normal opcode sequence length of Ramnit malware family tests arrives at 36,000. Be that as it may, the performance of LSTM is decreased quick when the length of input sequence surpasses 200. So how to process long sequence with LSTM network is critical.

One straightforward methodology for long sequences processing with LSTM network is Truncating And Padding (TAP). Specifically, TAP frst sets a fixed length N, truncates and disposes of the piece of long sequences exceeding length N, and cushions short sequences to length N with predefined identifier. It is convenient yet it abandons a great deal of information because of the truncation operation. It enhances the computational efficiency by sacrificing a little piece of the precision comparing with standard BPTT (or full BPTT) since standard BPTT calculation is less effective when backpropagation distance is excessively long. In addition, truncated BPTT is progressively reasonable for online learning, as it can rapidly adjust to the newly generated piece of an extremely long sequence. Generally speaking, truncated BPTT is reasonable and effective since it learns all sequence information comparing with TAP methodology.

`

In our scenario, a handy implement based on truncated BPTT calculation for LSTM network. Since gradients are only spread in the window, we first gap an opcode sequence into numerous subsequence's, where the length of every subsequence rises to the window length of truncated BPTT. Ten for every subsequence we simply do a full BPTT which equivalents doing the truncated BPTT for the entire sequence with no intersection window division. Most importantly, this permits LSTM to train in equal on a bunch of subsequences. One of the serious issues with LSTM is that the recurrent structure confines it to train a sequence sequentially, which is inefficient. Be that as it may, with these subsequences, our LSTM training procedure can be multiple times quicker.

Experimental results

Selection Comparison

Testing with an proposed model validation produced the following results:

- Original ASM Keyword Counts (1018 features): logloss = 0.01
- 10% Best ASM Features (202 features): logloss = 0.0184
- 20% Best ASM (350 features): logloss = 0.0132
- 30% Best ASM Feature Statistics (550 features):

multiclass logloss = 0.0133, accuracy score = 0.9978

- 40% Best ASM with feature statistics:

multiclass logloss = 0.0115, accuracy score = 0.9976

Models	**Accuracy (%)**	**AUC**	**TPR (%)**	**FPR (%)**	**EER (%)**	**Training time (h)**	**Detection time (ms)**
Linear Regression	96.21	0.9847	91.65	0.1	3.95	4.01	9.03
Decision Tree	97.52	0.9899	95.48	0.1	2.68	4.33	5.20
Gradient Boosting	98.47	0.9981	97.87	0.1	1.47	2.61	1.81
(Proposed EMDT)	99.35	0.9999	99.51	0.1	0.41	1.03	0.09

`

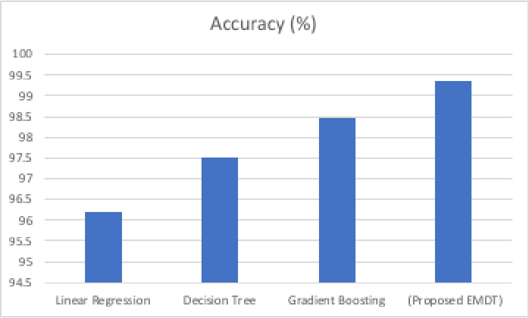

Figure 3-Accuracy comparison

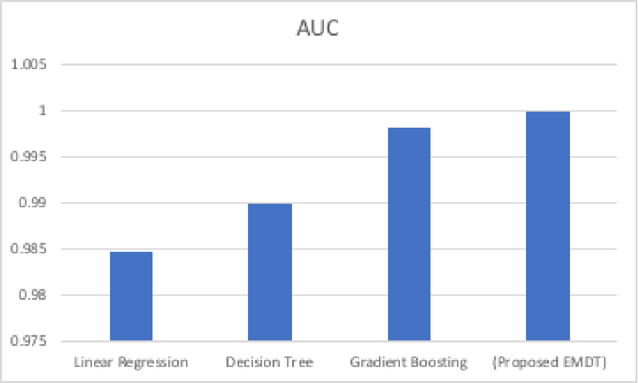

Figure 4-AUC comparison

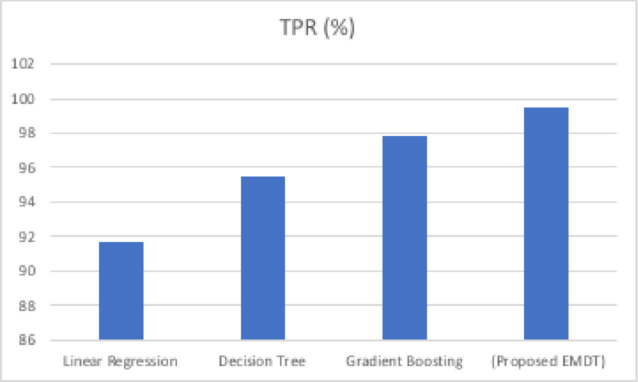

Figure 5-True Positive Ratio comparison

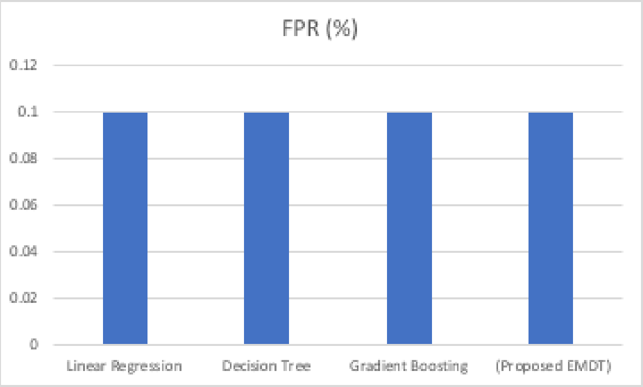

Figure 6-false Positive Ratio comparison

`

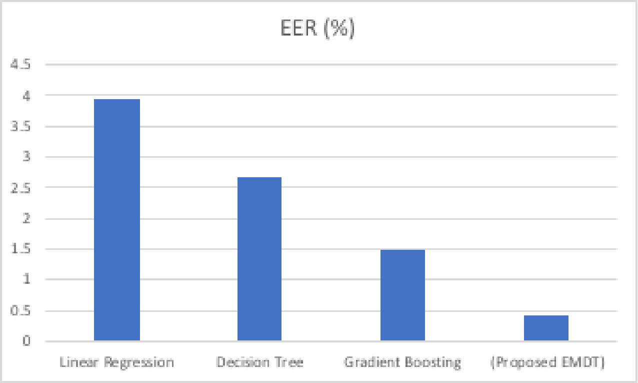

Figure 7-ERR Ratio comparison

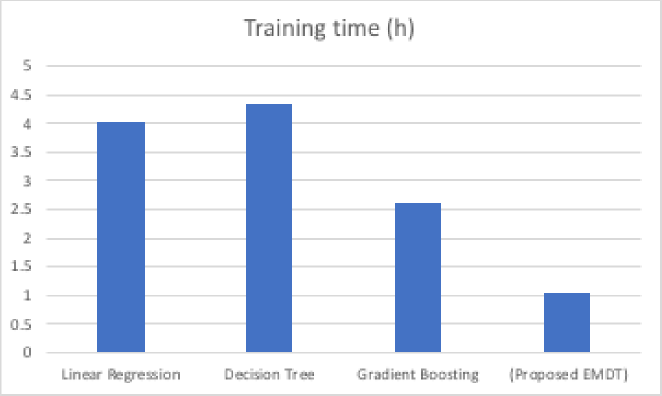

Figure 8-Training Time comparison

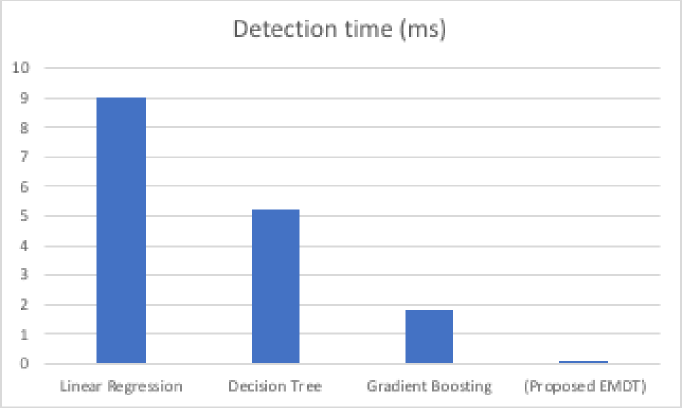

Figure 9-Detection Time comparison

Models	**Accuracy (%)**	**TPR (%)**
Linear Regression	96.21	91.65
Decision Tree	97.52	95.48
Gradient Boosting	98.47	97.87
(Proposed EMDT)	99.35	98.51

Methods	Accuracy (%)	Training time (h)	Detection time (s)
Linear Regression	96.21	21.35	10.54
Random Forest	96.85	20.65	8.01
Decision Tree	97.58	13.05	5.56
Adaboost	98.23	8.25	2.36
Gradient Boosting	97.36	2.64	1.87
(Proposed EMDT)	99.47	3.01	0.03

Conclusion

Some recent works attempt to avoid the detection of machine learning based malware classifiers by adversarial learning. Their experiments show that it is conceivable to generate ill-disposed examples based on a trained machine learning classifier. The centre of ill-disposed example crafting is to find a little perturbation on highlight vectors X of original malware test to change the classification results F to benign. Officially, they process the gradient of F as for to evaluate the direction wherein a perturbation σ in X would maximally change 's yield. The fundamental thought is shown in Figure 10. This assault scene is mainly brought about by the attributes of discriminative model and lacking of sufficient information. When dealing with a classification task with discriminative model, since it is practically difficult to have enough information to assist model with making decision in entire component space, discriminative model will attempt to expand the distance between tests and decision boundary for better classification result and, meanwhile, expand the region of every classification in highlight space. The benefit of this is to make the classification simpler; however the downside is that it additionally includes a ton of highlight spaces that don't obviously belong to current class, which enables aggressors generating antagonistic examples from this feature space.

REFERENCE

[1] A. Sung, J. Xu, P. C. and Mukkamala, S. (2004). Static Analyzer of Vicious Executables (SAVE). In Proceedings of the 20th Annual Computer Security Applications.

[2] Alazab, M., Venkataraman, S., and Watters, P. (2010). Towards Understanding Malware Behaviour by the Extraction of API calls. In 2nd CTC 2010 Ballarat (VIC), Australia.

[3] Anderson, B.and Quist, D., Neil, J., Storlie, C., and Lane, T. (2011). Graph Based Malware Detection Using Dynamic Analysis. In Journal in Computer Virology.

[4] AV-test (2017). Malware Statistics. http://www.av-test.org/en/statistics/

`

malware/.

[5] Bayer, U., Comparetti, P., Hlauschek, C., and Kruegel, C. (2009). Scalable, Behavior-Based Malware Clustering. In Proceedings of the 16th Annual Network and Distributed System Security Symposium.

[6] Bazrafshan, Z., Hashemi, H., Fard, S. M. H, and Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In Information and Knowledge Technology.

[7] Bengio, Y., Simard, P., and Frasconi., P. (1994). Learning long-term dependencies with gradient descent is di_cult. IEEE Transactions on Neural Networks.

[8] BooJoong Kang, Suleiman Y. Yerima, K. M. S. S. (2016). N-opcode Analysis for Android Malware Classi_cation and Categorization.

[9] Christodorescu, M., Jha, S., and Kruegel, C. (2007). Mining speci_cations of malicious behavior. In Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESECFSE).

[10] Clarify (2014). Convolutional Neural Networks. https://www.clarifai.com/ technology.

[11] Cohen, F. (1987). Computer Viruses: Theory and Experiments. http://web.eecs. umich.edu/~aprakash/eecs588/handouts/cohen-viruses.html.

[12] Conti, G., Bratus, S., Sangster, B., Ragsdale, R., Supan, M., Lichtenberg, A., Perez-Alemany, R., and Shubina, A. (2010). Automated Mapping of Large Binary Objects Using Primitive Fragment Type Classi_cation. In The proceedings of The Digital Forensic Research Conference DFRWS.

[13] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. (2009). The WEKA Data Mining Software. In ACM SIGKDD Explorations Newsletter.

[14] Han, K. S., Lim, J. H., and Im, E. G. (2013). Malware analysis method using visualization of binary _les. In Proceedings of Research in Adaptive and Convergent Systems ACM.

[15] He, Kaiming; Zhang, X. R. S. S. J. (2015). Deep Residual Learning for Image Recognition. eprint arXiv:1512.03385, ARXIV.

[16] Helfman, J. (1995). Dotplot patterns: A literal look at pattern languages. TAPOS, 2:31{41.

[17] Honkela, A. (2001). Nonlinear switching state-space models. URLhttp://www.hiit. fi/u/ahonkela/dippa/dippa.html.

[18] Hubel, D. and Wiesel, T. (1968). Receptive _elds and functional architecture of monkey striate cortex. Journal of Physiology (London).

[19] ICT (2016). ICT: Facts and Figures. http://www.itu.int/en/ITU-D/Statistics/

Documents/facts/ICTFactsFigures2016.pdf.

[20] Indyk, P. and Motwani, R. (1998). Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In Proceedings of 30th Annual ACM Symposium on Theory of Computing, Dallas.

[21] Islam, R., Tian, R., Batten, L. M., and Versteeg, S. (2013). Classi_cation of malware based on integrated static and dynamic features. In Journal of Network and Computer Applications.

[22] Kaspersky (2014). Cybercrime, Inc.: How pro_table is the business? https://blog. kaspersky.com/cybercrime-inc-how-profitable-is-the-business/15034/.

[23] Ki, Y., Kim, E., and Kim, H. K. (2015). A Novel Approach to Detect Malware Based on API Call Sequence Analysis. International Journal of Distributed Sensor Networks.

[24] Kohonen, T. (1995). Self-Organizing Maps. Springer.

[25] Kolbitsch, C. (2011). Anubis. https://seclab.cs.ucsb.edu/academic/projects/ projects/anubis/.

[26] Kolter, J. and Maloof, M. (2004). Learning to detect malicious executables in the wild. In In Proc. of the 10th ACM Int. Conf. on Knowledge Discovery and Data Mining.

[27] Kong D, Y. G. (2013). Discriminant malware distance learning on structural information for automated malware classi_cation. In Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. ACM.

[28] LIU, L., Bao-sheng WANG, YU, B., and Qiu-xi ZHONG (2016). Automatic Malware Classi_cation and New Malware Detection using Machine Learning. In Frontiers of Information Technology and Electronic Engineering.

[29] M. Schultz, M. Eskin, E. Z. and Stolfo, F. (2001). Data Mining Methods for Detection of New Malicious Executables. In In Proc. of the 22nd IEEE Symposium on Security and Privacy.

[30] Makandar, A. and Patrot, A. (2015). Malware Analysis and Classi_cation using Arti_cial Neural Network. In Trends in Automation Communications and Computing Technology.

[31] Malshare (2012). Malware Repository. http://malshare.com/.

[32] Microsoft (2017). Naming malware. https://www.microsoft.com/en-us/ security/portal/mmpc/shared/malwarenaming.aspx.

[33] Montufa, G., Pascanu, R., Cho, K., and Bengio., Y. (2014). Learning long-term dependencies with gradient descent is di_cult on the number of linear regions of deep neural networks. NIPS.

[34] Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of Static Analysis for Malware Detection. In IEEE Computer Society.

[35] Nari, S. and Ghorbani, A. (2013). Automated Malware Classi_cation Based on Net-

`

work Behaviour. In Proceedings of International Conference on Computing, Networking and Communications (ICNC), San Diego.

[36] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011a). Malimg Dataset. http://old.vision.ece.ucsb.edu/spam/malimg.shtml.

[37] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011b). Malware Images: Visualization and Automatic Classi_cation. In Proceedings of International Symposium on Visualization for Cyber Security.

[38] Ollmann, G. (2008). The Evolution of commercial malware development kits and colour-by-numbers custom malware. In Computer Fraud and Security.

[39] Pandalabs (2016). Quaterly Report. http://www.pandasecurity.com/ mediacenter/src/uploads/2016/05/Pandalabs-2016-T1-EN-LR.pdf.

[40] Pandalabs (2017). Quaterly Report. http://www.pandasecurity.com/ mediacenter/src/uploads/2017/05/Pandalabs-2017-T1-EN.pdf.

[41] Peter, E. and Schiller, T. (2008). A practical guide to honeypots. http://www.cs. wustl.edu/~jain/cse571-09/ftp/honey.pdf.

[42] R. Tian, L. M. B. and Versteeg, S. C. (2008). Function length as a tool for malware classi_cation. In In Proc. of the 3rd Int. Conf. on Malicious and Unwanted Software.

[43] Radu, S. P., Hansen, S. S., Larsen, Thor. M. T., Stevanovic, M., Pedersen, J. M., and Czech, A. (2015). Analysis of malware behavior: Type classi_cation using machine learning. In CyberSA.

[44] Raghakot (2015). ResNet. https://github.com/raghakot/keras-resnet.

[45] Rieck, K., Holz, T., Willems, C., Dussel, P., and Laskov, P. (2008). Learning and classi_cation of Malware behaviour. In Detection of Intrusions and Malware, and Vulnerability Assessment.

[46] S. Staniford, V. P. and Weaver, N. (2002). How to own the internet in your spare time. In Proceedings of the 11th USENIX Security Symposium.

[47] Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. (2012). OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection.

[48] Santos, I., Nieves, J., and Bringas, P. (2011). Semi-Supervised Learning for Unknown Malware Detection. In Symposium on Distributed Computing and Arti_cial Intelligence Advances in Intelligent and Soft Computing.

[49] Schultz, M. G., Eskin, E., and Zadok, F. (2001). Data Mining Methods for Detection of New Malicious Executables. In In Proc. of the 22nd IEEE Symposium on Security and Privacy.

[50] Siddiqui, M. and Wang, M. C. (2009). Detecting Internet Worms Using Data Mining Techniques. In Journal of Systemics, Cybernetics and Informatics.

[51] SIKORSKI M., H. A. (2012). Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.

[52] Spa_ord, E. H. (1989). The Internet worm incident. In Proceedings of the 2nd European Software Engineering Conference. 446?468.

[53] Szor, P. (2005). The Art of Computer Virus Research and Defense.

[54] Torralba, A., Murphy, K., Freeman, W., and Rubin, M. (2003). Classi_cation of malware based on integrated static and dynamic features. In In Proceedings of ICCV.

[55] Vinod, P., Jaipur, R., Laxmi, V., and Gaur, M. (2009). Survey on malware detection methods. In Proceedings of the 3rd Hacker's Workshop on Computer and Internet Security (IITKHACK?09).

[56] VirusShare (2011). Malware Repository. https://virusshare.com/.

[57] VirusTotal (2004). Online Malware Report Generator. https://www.virustotal.com/.

[58] VirusTotal (2017). Daily Statistics. https://www.virustotal.com/en/statistics/.

[59] Wagener, G., State, R., and Dulaunoy, A. (2008). Malware behaviour analysis. Journal in Computer Virology. In Proceedings of the 5th International Conference on Malicious and Unwanted Software 2010.

[60] Wikipedia (2016). Mirai Malware. https://en.wikipedia.org/wiki/Mirai_(malware).

[61] Wikipedia (2017a). Ransomware. https://en.wikipedia.org/wiki/Ransomware.

[62] Wikipedia (2017b). WannaCry Ransomware. https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.

[63] Willems, C., Holz, T., and Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using Cwsandbox. In IEEE Security and Privacy.

[64] Yoo, I. S. (2004). Visualizing windows executable virus using self-organizing maps. In Proceedings of ACM workshop on Visualization and data mining for computer security.

[65] Zeiler, Matthew D; Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. eprint arXiv:1311.2901S.

[66] Zhang, Q. and Reeves, D. (2007). Metaware: Identifying Metamorphic Malware. In Proceedings of the 23rd Annual Computer Security Applications Conference.

[67] Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Scholkopf, B. (2003). Learning with local and global consistency. In Advances in Neural Information Processing Systems 16: Proceedings of the 2003.